DRANKULA: A McEliece-like Rank Metric based Cryptosystem Implementation

Ameera Salem Al Abdouli, Mohamed Al Ali, Emanuele Bellini, Florian Caullery, Alexandros Hasikos, Marc Manzano and Victor Mateu
DarkMatter, U.A.E.

Keywords: Code-based Cryptography, Public Key Cryptography, Cryptosystem, Software Implementation, Post-quantum Cryptography.

Abstract: We present and analyze the performance of DRANKULA, a McEliece-like cryptosystem implementation using *rank metric* instead of Hamming distance. Namely, we use the scheme proposed by Loidreau in PQCrypto 2017 using Gabidulin codes. We propose a set of carefully selected parameters and we address several non-trivial issues when porting this scheme into real-world systems as, for example, the generation of errors of a given rank. We provide the pseudo-code of the core algorithms of the cryptosystem. In addition, we also show code optimization when special instructions like Carry-less multiplications are available. Moreover, we argue how to have a practical and side-channel resistant version of the cryptosystem. We integrated the scheme in Open Quantum Safe and benchmarked it against the other schemes implemented there. Our results show that DRANKULA can be a practical alternative to other well-known quantum-safe schemes.

1 INTRODUCTION

Substantial advances in quantum computing in the past decade have re-assured the scientific community about the necessity to build quantum-resistant cryptosystems (Devoret and Schoelkopf, 2013). Recently, Google announced Bristlecone, a new 72 qubits quantum processor (Google, 2018). Post-Quantum Cryptography (PQC) has raised as the preferred solution to face the threat that quantum computers pose to traditional public-key cryptography based on number theory (Shor, 1997). The recent announcement by the National Institute of Standards and Technology (NIST) to define new standards for public-key encryption, digital signatures and key-exchange schemes has only augmented the attention towards PQC (Chen et al., 2016).

There exist several alternative problems to classical public-key cryptography, which is based on integer factorization or discrete logarithms. Lattice-based cryptography, multivariate cryptography, hash-based cryptography schemes, isogeny-based cryptography and code-based cryptography can be used to design cryptosystems secure against both classical and quantum computers (Bernstein et al., 2008), and are thus regarded as PQC algorithms, being candidates for becoming the next standards defined by NIST.

Code-based cryptography is the oldest PQC family known and consequently is the most thoroughly studied among the PQC candidates (Sendrier, 2017). McEliece proposed the first code-based public-key cryptosystem (McEliece, 1978), which is based on binary Goppa codes, and so far has withstood all cryptanalytic efforts. Another well known cryptosystem was proposed by Niederreiter in (Niederreiter, 1986). Recently, Bernstein, Chou and Schwabe presented McBits, which improved decryption time with respect to Niederreiter (Bernstein et al., 2015). An improved version was presented in (Chou, 2017). However, the usage of the aforementioned schemes in real world applications has been prohibitive because of the significantly large key sizes (e.g., McBits requires 64KB to achieve 2^{80} classical security level). Therefore, the most important challenge that code-based cryptography faces is how to reduce key sizes in order to be implemented on limited resource devices.

Two different lines have been investigated with the purpose of reducing key sizes. On the one hand, several works have tried to address this drawback by designing McEliece variants with more compact keys by using quasicyclic codes (Gaborit, 2005; Misoczki et al., 2013; Chou, 2016). On the other hand, the usage of rank metric rather than Hamming distance has been considered.

The rank metric was introduced by Gabidulin in (Gabidulin, 1985) and he proposed a family of codes, the Gabidulin codes, equivalent to Reed-Solomon codes in Hamming distance, which can be decoded in

Abdouli, A., Ali, M., Bellini, E., Caullery, F., Hasikos, A., Manzano, M. and Mateu, V.
DRANKULA: A McEliece-like Rank Metric based Cryptosystem Implementation.
DOI: 10.5220/0006838100640075
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 64-75
ISBN: 978-989-758-319-3

polynomial time. Rank Syndrome Decoding (RSD) is the equivalent in rank metric of the Syndrome Decoding problem in Hamming distance. Later on, Gabidulin et al. proposed the GPT cryptosystem, a McEliece-like cryptosystem based on Rank metric (Gabidulin et al., 1991). One of the advantages of using Gabidulin codes is that the complexity of the best known attacks for solving the RSD problem have an exponential complexity which is quadratic in the parameters of the system. As a consequence, key sizes can be notably reduced to achieve the same security level than a Hamming distance-based cryptosystem.

Then, Overbeck proposed a framework that devastated all Gabidulin codes based encryption schemes (Overbeck, 2005; Overbeck, 2008). Furthermore, evolutions of the GPT cryptosystem have been shown to be also breakable in polynomial-time (Otmani et al., 2016). Nevertheless, Loidreau has proposed a new McEliece-like rank metric based encryption scheme from Gabidulin codes that is not affected by the so-called Overbeck's attacks (Loidreau, 2017).

In this work we carry out the first implementation of the scheme proposed in (Loidreau, 2017), which we have called DRANKULA. We present a set of selected parameters that allow the scheme to achieve different levels of quantum security; namely 64, 96 and 128 bit. We motivate this parameter selection by implementation constraints and optimization possibilities. Moreover, we go through all the caveats of the implementation of a cryptosystem based on rank metric codes, and we provide the pseudo-code for the main algorithms. We also discuss several countermeasures that would make the implementation side-channel resistant.

1.1 Organization of the Paper

The rest of the paper is structured as follows. First, in Section 2 we outline all the code-based Key Encapsulation Mechanism (KEM) and Public Key Encryption (PKE) submissions to the NIST competition and then in Section 3 we introduce the cryptosystem we are implementing on this paper, and the selected parameters. Section 4 is devoted to the base field arithmetic and matrix operations that are required to implement for running the cryptosystem. In Section 5 we present the necessary code-blocks used to build the cryptosystem, namely, (a) the key generation, (b) the encryption which includes the error generation, and (c) the decryption. Section 6 gives an overview of the performance of the scheme compared with other post-quantum cryptography schemes. Section 7 presents some directions to have a side-channel resistant implementation. Finally, we provide the conclusions of our proposal in Section 8.

2 RANK-METRIC NIST SUBMISSIONS

There were four code-based KEM and PKE schemes based on the rank metric submitted to the first round of the NIST post-quantum cryptography standardization competition (NIST, 2018). First, Ouroboros-R (Deneuville et al., 2017), is a cryptosystem that inherits features from MDPC-McEliece (Misoczki et al., 2013) and the rank metric quasi-cyclic family. The security relies on decoding small weight vectors of random quasi-cyclic codes. The advantages are the security reduction with small public key sizes resulting from cyclicity, the easy evaluation of the low decryption failure probabilities and the efficiency of the decoding compared to MDP. Second, LAKE (Gaborit et al., 2013), which stands for *Low rAnk parity check codes Key Exchange*, is a rank-metric scheme for which the authors provide an efficient probabilistic decoding algorithm. The quasi-cyclic family of codes that is being used produces a public key which is significantly smaller than MDPC codes. Similar to LAKE, LOCKER (Aragon et al., 2017) is built on a small variation of low rank parity check codes and has been adapted to support low decryption probability failures with a versatile choice of parameters. Considering the low decryption failure probabilities the scheme is efficient in terms of key sizes and computational complexity. These three schemes share the property of a probabilistic decryption algorithms. In contrast, DRANKULA offers a deterministic decryption procedure which makes it more suitable in scenarios where we cannot tolerate decryption failures.

Last, *RQC* (Melchor et al., 2016), is another rank-metric scheme that is also based on the difficulty of decoding random quasi-cyclic codes. The key-size of the scheme is restricted to a few thousands of bytes and has a deterministic decoding algorithm. It is also resistant to code hidden structure recovery attacks. Although having a bigger key size, DRANKULA reduces the ciphertext expansion by a factor of 2 in comparison to RQC.

3 CRYPTOSYSTEM DESCRIPTION

The scheme we have chosen to implement is the one proposed in (Loidreau, 2017). It is a scheme based on Gabidulin codes using a special sub-space for the entries of the scrambling matrix which transforms the private key into the public key.

The essential difference between the presented

cryptosystem and traditional McEliece instantiations is that, instead of XORing the encoded plaintext with errors of a given Hamming weight, we do it with an error of a specific *rank weight*.

Definition 1. *Let $e = \{e_1, \ldots, e_n\} \in \mathbb{F}_{2^m}^n$. The rank weight of e is defined by*

$$\mathrm{rk}\left(\begin{bmatrix} e_{1,1} & \cdots & e_{n,1} \\ e_{1,2} & \cdots & e_{n,2} \\ \vdots & \ddots & \vdots \\ e_{1,m} & \cdots & e_{n,m} \end{bmatrix}\right)$$

where $e_{i,j}$ is the jth component of e_i seen as a vector over $\mathbb{F}_2$.

Definition 2 (Gabidulin codes)**.** *Let $k < n \leq m$ be non-negative integers and let $\{g_1, ..., g_n\} \in \mathbb{F}_{2^m}$, be linearly independent over $\mathbb{F}_2$. Let $[i] = 2^i$ such that $x \to x^{[i]}$ is the ith power of the Frobenius automorphism $x \to x^2$. The code $Gab_{k,n}(\boldsymbol{g})$, is the linear code with generator matrix*

$$G = \begin{bmatrix} g_1 & \cdots & g_n \\ g_1^{[1]} & \cdots & g_n^{[1]} \\ \vdots & \ddots & \vdots \\ g_1^{[k-1]} & \cdots & g_n^{[k-1]} \end{bmatrix}$$

that is:

$$Gab_{k,n}(\boldsymbol{g}) = \{\boldsymbol{xG} | \boldsymbol{x} \in \mathbb{F}_{2^m}^k\}.$$

These codes can be decoded in polynomial-time for errors of rank weight up to $\lfloor (n-k)/2 \rfloor$ (Gabidulin, 1985).

Gabidulin Codes are the rank-metric equivalent of Reed Solomon Codes. The scheme's approach is to choose a randomly selected vector space of $\mathbb{F}_2^m$ of fixed dimension which is used to scramble the codes. The proof of correctness of the cryptosystem is based on the rank multiplication property, the same one used to show that the Low Rank Parity Check (LRPC) codes decoding procedure works.

The scheme has four parameters:

- m:= the degree of the extension of $\mathbb{F}_2$ in which our code will be defined
- n:= the length of the code
- k:= the dimension of the code
- λ:= the dimension of a randomly selected subspace of $\mathbb{F}_{2^m}$

The security bounds of the scheme given in (Loidreau, 2017) are the following:

- Decoding of the ciphertext in the public code corresponds to the complexity of solving Bounded Distance binary Rank decoding BDR problem which is NP-hard. In that setting, the decoding complexity is equal to $m^3 2^{(\lambda \mathrm{Rk}(e)-1)\lfloor (k \min(m,n))/n \rfloor}$ binary operations for a classical computer and to $m^3 2^{\frac{1}{2}(\lambda \mathrm{Rk}(e)-1)\lfloor (k \min(m,n))/n \rfloor}$ operations for a quantum computer.
- The lower bound on the complexity of distinguishability of the public code from a random code is $2^{(\lambda-1)m-(\lambda-1)^2}$.

Finally, the scheme is separated into the three following parts:

- **Key Generation:**
  - **Private Key:**
    1. Select a Gabidulin code of length n and dimension k over $\mathbb{F}_{2^m}$ with generator matrix G_{priv}
    2. Randomly generate a non-singular $k \times k$ matrix $S \in \mathcal{M}_k(\mathbb{F}_{2^m})$
    3. Randomly select a vectorial subspace $V \subset \mathbb{F}_2^m$ of dimension λ
    4. Randomly generate a non-singular $P \in \mathcal{M}_n(V)$
  - **Public Key:** The public code C_{pub} has $G_{pub} = SG_{priv}P^{-1}$ as a generator matrix.
- **Encryption:**
  1. Choose a random vector $e \in \mathbb{F}_{2^m}^n$ of rank weight $t := \lfloor (n-k)/(2\lambda) \rfloor$
  2. Compute $y = xG_{pub} + e$
  3. Send y
- **Decryption:**
  1. Compute $yP = xSG + eP$
  2. Recover xS and eP by decoding and recover x by multiplying xS by S^{-1}

We refer to (Loidreau, 2017) for the proof of work.

3.1 Parameters Selection

The parameters proposed in (Loidreau, 2017) were not optimal for its implementation because they were focused on reducing the size of both private and public keys instead. For our implementation, we have chosen a base field which is either $\mathbb{F}_{2^{64}}$ or $\mathbb{F}_{2^{96}}$ to fit nicely into the words structure available on modern machines and carry out optimizations based on special sets of instructions (e.g. carry-less multiplication). We targeted 3 levels of post-quantum security, namely 64 bit, 96 bit and 128 bit and we have determined the following parameters. Table 1 shows our parameter sets and their respective security levels. Such values are computed following the formulas given in (Loidreau, 2017) and are consistent with the ones presented in that paper.

Table 1: DRANKULA security parameters.

m	n	k	λ	t	dec. cplx	quantum dec. cplx.	key recovery cplx.	public key size
64	63	31	3	5	142	78	124	7.75KB
96	71	35	3	6	194.76	104.76	188	14.77KB
96	96	48	4	6	259.75	139.75	279	27KB

4 IMPLEMENTATION OF CORE COMPONENTS

Given the nature of the cryptosystem, the core components are the field arithmetic and the matrix operations. Therefore, the performance of the resulting algorithms will heavily rely on how efficient such operations are.

4.1 Field Arithmetic

We implemented finite field arithmetic for the two binary fields $\mathbb{F}_{2^m}$, with $m = 64$ and $m = 96$, representing elements as polynomials of degree $m-1$. As it was proven in Swan's Theorem (Swan, 1962) on factorization of polynomials over finite fields, given that 64 and 96 are multiples of 8 there is no irreducible trinomial. Therefore, we used the two following irreducible pentanomial $f_{64}(x) = x^{64} + x^4 + x^3 + x + 1$ and $f_{96}(x) = x^{96} + x^{10} + x^9 + x^6 + 1$, respectively, both provided by the Allan Steel database incorporated in Magma software (Bosma et al., 1997). Notice that in the case of f_{64} the pentanomial has also lowest possible intermediate degree, allowing the shortest shift during the reduction operations. Such irreducible polynomial for the case of $\mathbb{F}_{96}$ has 7 terms, i.e. $x^{96} + x^6 + x^5 + x^3 + x^2 + x + 1$, which is not convenient to use compared with the pentanomial.

We stored elements in both fields using 128 unsigned integers, with unused bits set to zero.

Addition and subtraction of two elements are a simple XOR operation.

The multiplication of two polynomials has been performed using PCLMULQDQ instruction (Gueron and Kounavis, 2010). In the case of $\mathbb{F}_{96}$, three calls to such instruction are needed, since the instruction operates on 64 bit inputs. For this reason we split one element of 96 bits in two elements of 32 and 64 bits respectively, and then apply Karatsuba-Ofman method (Karatsuba and Ofman, 1962). In the case of $\mathbb{F}_{64}$, only one call to the instruction is needed, and there is no need to split the element.

Since the multiplication provides polynomials of degree at most $2m-2$, we need to perform the reduction of such result. The two algorithms for reduction are presented in Algorithm 1 and Algorithm 2, where the symbols $\ll, \gg$ denote field multiplication and division by x respectively (left and right shift operators), $\oplus$ is the field addition (XOR operator), and $||$ is the concatenation operator.

Algorithm 1: Reduction in $\mathbb{F}_{2^{64}}$.

input : $A = (a_{126}, \ldots, a_0) \in \mathbb{F}_2^{127}$
output : $C = A \mod f_{64} \in \mathbb{F}_2^{64}$

$A_0 = (a_{63}, \ldots, a_0) \in \mathbb{F}_2^{64}$
$A_1 = (0, a_{126}, \ldots, a_{64}) \in \mathbb{F}_2^{64}$
$A_0 = A_0 \oplus (A_1 \ll 4) \oplus (A_1 \ll 3) \oplus (A_1 \ll 1) \oplus A_1$
$A_1 = (A_1 \gg 60) \oplus (A_1 \gg 61) \oplus (A_1 \gg 63)$
$T = (0, \ldots, 0, a_{67}, a_{66}, a_{65}, a_{64}) \in \mathbb{F}_2^{64}$
$A_0 = A_0 \oplus (T \ll 4) \oplus (T \ll 3) \oplus (T \ll 1) \oplus T$
return A_0

Algorithm 2: Reduction in $\mathbb{F}_{2^{96}}$.

input : $A = (a_{190}, \ldots, a_0) \in \mathbb{F}_2^{191}$
output : $C = A \mod f_{96} \in \mathbb{F}_2^{96}$

$A_0 = (a_{63}, \ldots, a_0) \in \mathbb{F}_2^{64}$
$A_1 = (a_{127}, \ldots, a_{64}) \in \mathbb{F}_2^{64}$
$A_2 = (0, \ldots, 0, a_{190}, \ldots, a_{128}) \in \mathbb{F}_2^{64}$
$A_0 = A_0 \oplus (A_2 \ll 42) \oplus (A_2 \ll 41) \oplus (A_2 \ll 38) \oplus (A_2 \ll 32)$
$A_1 = A_1 \oplus (A_2 \gg 22) \oplus (A_2 \gg 23) \oplus (A_2 \gg 26) \oplus (A_2 \gg 32)$
$T = (0, \ldots, 0, a_{127}, \ldots, a_{96}) \in \mathbb{F}_2^{64}$
$A_0 = A_0 \oplus (T \ll 10) \oplus (T \ll 9) \oplus (T \ll 6) \oplus T$
return $A_1 || A_0 \in \mathbb{F}_2^{96}$

To limit memory usage, the inversion on the field relies on the Extended Euclidean Algorithm (Hankerson et al., 2006). An alternative requiring more memory, which we did not consider, would be Itoh-Tsujii algorithm with precomputed powers (Itoh and Tsujii, 1988).

4.2 Matrix Operations

The operations needed in our implementation are the following:

- Matrix multiplications
- Gaussian elimination in order to perform:
  - Matrix inversion for square matrices
  - Assertion of linear independence
  - Reduction of the generator matrix of the public code to the systematic form (i.e. rewrite G_{pub} as $[Id_k||G'_{pub}]$)

It has to be noted that those operations were implemented with two variants. That is, one for the matrices with coefficients in $\mathbb{F}_2$ and one for the matrices with coefficients in $\mathbb{F}_{2^m}$. As the sizes of the matrices are quite small (96×96 at maximum), the naive algorithms were actually faster than any optimized version. For example, the matrix multiplication is not using Strassen's algorithm and matrix inversion is carried out using Gaussian elimination.

5 ALGORITHMS IMPLEMENTATION

Loidreau's proposed scheme consists of the three algorithms sumarized in Section 3. Here we present our implementation for each of them.

5.1 Key Generation

The key generation is pretty simple. It just requires to generate random matrices with coefficients in $\mathbb{F}_2$ or $\mathbb{F}_{2^m}$ of the right dimension and to ensure that they respect the conditions given in the scheme specification. If the randomly generated matrix does not fulfill the conditions, then repeat the process until one suitable candidate is found.

More precisely, the conditions are the following:

- The generators $g_1, \ldots, g_n$ of the Gabidulin code with generator matrix G_{priv} are all linearly independent over $\mathbb{F}_2$. That is, the binary matrix

$$\begin{bmatrix} g_{1,1} & \cdots & g_{n,1} \\ \vdots & \vdots & \vdots \\ g_{1,m} & \cdots & g_{n,m} \end{bmatrix},$$

  where $g_{i,j}$ is the j-th bit of g_i, has a rank equal to n.
- The λ random elements of $\mathbb{F}_{2^m}$ which shall be seen as the basis of V have to be linearly independent over $\mathbb{F}_2$.
- Matrices $P \in \mathcal{M}_n(V)$ and $S \in \mathcal{M}_k(\mathbb{F}_{2^m})$ should be non-singular.

Luckily, the density of non-singular matrices with coefficients in finite field is high (see (Maples, 2013)) and the sample-rejection strategy is guaranteed to be fast for the generation of P and S or even of $g_1, \ldots, g_n$ in the case where $m = n$.

5.2 Encryption

The encryption operation uses a matrix multiplication and a vector addition. Nevertheless, the generation of an error vector of a given rank is a tricky process. Indeed, the probability that a randomly generated $n \times m$ matrix with coefficients in $\mathbb{F}_2$ has a rank equal to the value specified in the scheme is low (see (Maples, 2013)). There is a need for a more optimized strategy. The idea is to generate t random linearly independent vectors of $\mathbb{F}_2^m$ and then generate n random linear combinations of those vectors. This is the method which is also used in (Deneuville et al., 2017). The pseudo-code of the error generation is given in Algorithm 3.

Algorithm 3: Random error generation.

input : The binary field size m, the length of the error n and the error rank t
output : A random element $e := \{e_1, \ldots, e_n\} \in \mathbb{F}_{2^m}^n$ such that $\mathrm{rk}(e) = t$

```
while not_linearly_independent(b_1, ..., b_t) do
    b_1, ..., b_t ← generate_random_vectors(t, F_{2^m})
while rk(e) ≠ t do
    for i from 1 to n do
        rand ←$ {0,1}^t
        e_i = Σ_{k=1}^{t} rand_k × b_k
return e
```

Even if highly unlikely, it could happen that the coefficient associated to one (or more) of the generating vectors is always zero. In that case, the error vector would have a rank lower than t. This is why the rank of the output error vector must be checked and, in case it does not fulfill the requirements of the scheme, another error vector must be generated.

5.3 Decryption

Given an encrypted message $y = xSG_{priv}P^{-1} + e$, the first step to decrypt it is to multiply y by P to obtain $yP = xSG_{priv} + eP$. After that, the Algorithm 11

might be used to obtain the corrected message which is equal to xS. Finally, the cleartext x is computed by multiplying xS and S^{-1}. Notice that S is only needed during the public key generation but, for decryption, it is S^{-1} the required matrix. Therefore, in our implementation we store S^{-1} as part of the private key instead of S in order to avoid the computations and memory consumption from the inversion in each decryption operation.

The core component of the decryption algorithm is the error detection and correction. In order to optimize this step, we use the algorithms provided in (Puchinger and Wachter-Zeh, 2015) and (Wachter-Zeh, 2013). Here we provide the pseudo-code of the algorithms. We refer to the aforementioned papers for the proof that these algorithms correctly decode any error with rank less or equal to $\lfloor (n-k)/2\lambda \rfloor$, and an in-depth explanation of the main idea. Essentially, this decoding procedure can be seen as the equivalent of Gao's algorithm (Gao, 2003) for decoding Gabidulin codes. It outputs an evaluation polynomial of the estimated message thus avoiding to solve large linear equations systems depending on the syndrome. To the best of our knowledge, this is the fastest decoding procedure for Gabidulin codes. The algorithm takes advantage of the properties of linearized polynomials to achieve a sub-quadratic complexity. Since linearized polynomials are used in all the algorithms for decryption, we briefly recall their definition.

Definition 3 (Linearized polynomials)**.** *A linearized polynomial $a(x) \in \mathbb{F}_{2^m}[x]$ is a polynomial of the form*

$$\sum_{i=0}^{d} a_i x^{[i]},$$

where $x^{[i]} = x^{2^i}$ and d is the 2-degree of a, denoted as $\deg_2(a)$ in this paper.

We will denote by $\mathbb{L}_{2^m}[x]$ the space of all linearized polynomials of $\mathbb{F}_{2^m}[x]$ and by $\mathbb{L}_{2^m}[x]_{\leq s}$ the space of all linearized polynomials of $\mathbb{F}_{2^m}[x]$ with maximum degree s. When embedded with the standard addition $+$ and composition $\circ$, $\mathbb{L}_{2^m}[x]$ is a left (right) euclidean ring. Meaning that we can define a left (right) division.

5.3.1 Algorithms Notation

Given that the algorithms required to compute the decoding of a message are presented next, here we introduce the notation rules we have used:

- We denote as c_i the coefficient at position i from polynomial c.
- We denote as $\text{Frobenius}(x,n)$ the computation of the n-th iteration of Frobenius automorphism over x.
- Given a linearized polynomial p, we denote as $p = 0$ the assignment to p of a polynomial with all its coefficients being 0.
- Given a linearized polynomial p, we denote as $p = x$ to represent that p is the linearized polynomial with coefficient 1 in the lowest degree position (i.e. the coefficient of the term x) and 0s in all others.
- We use the notation $U[i]$ to denote that U is a vector of elements in $\mathbb{F}_{2^m}$ where we access its ith position.
- We use —— as the concatenation operation.
- We use $\#U$ to denote the amount of elements of a vector U.

5.3.2 Linearized Polynomial Multiplication

The first needed subroutine is the fast multiplication of two linearized polynomials. Recall that the multiplication in the ring $\mathbb{L}_{2^m}[x]$ is actually the standard composition. The details of its implementation are in Algorithm 4.

Algorithm 4: Multiplication.

input : $a, b \in \mathbb{L}_{2^m}[x]_{\leq s}$
output: $c = a \circ b \in \mathbb{L}_{2^m}[x]_{\leq s}$
for *i from* 0 *to* $\deg_2(a) + \deg_2(b)$ **do**
  $y = 0$
  for *j from* 0 *to i* **do**
    if $j \leq \deg_2(a)$ *and* $i - j \leq \deg_2(b)$ **then**
      $z = \text{Frobenius}(b_{i-j}, j)$
      $z = z \cdot a_j$
      $y = y + z$
  $c_i = y$
return c

5.3.3 Linearized Polynomial Divisions

As $\mathbb{L}_{2^m}[x]$ is a Euclidean ring, we can define a division. As the multiplication in that ring is not commutative, there exist a well-defined left and right division. The fast algorithms for the left and right division are given by Algorithms 5 and 6, respectively.

5.3.4 Right Linearized Extendend Euclidean Algorithm

We can use a Linearized version of the Euclidean Algorithm (LEEA) based on the divisions defined

Algorithm 5: LeftDivision.

input : $a, b \in \mathbb{L}_{2^m}[x] - \{0\}$
output : $q, r \in \mathbb{L}_{2^m}[x]$ such that $b \circ q + r = a$
initialize: $r = a, \quad db = \deg_2(b), \quad q = 0, \quad di = \deg_2(r),$

if $di < db$ **then**
  return q, r
while $di \geq db$ **do**
  $q' = 0$
  $q'_{di-db} = \text{Frobenius}(r_{di}/b_{db}, m - db)$
  $q = q + q'$
  $r = r + b \circ q'$
  $di = \deg_2(r)$
return q, r

Algorithm 6: RightDivision.

input : $a, b \in \mathbb{L}_{2^m}[x] - \{0\}$
output : $q, r \in \mathbb{L}_{2^m}[x]$ such that $q \circ b + r = a$
initialize: $r = a, \quad db = \deg_2(b), \quad q = 0, \quad di = \deg_2(r),$

if $di < db$ **then**
  return q, r
while $di \geq db$ **do**
  $q' = 0$
  $q'_{di-db} = r_{di} / \text{Frobenius}(b_{db}, di - db)$
  $q = q + q'$
  $r = r + q' \circ b$
  $di = \deg_2(r)$
return q, r

above. Given that only the version using right division is required, in Algorithm 7 we show the implementation of the right LEEA algorithm.

5.3.5 Minimal Subspace Polynomial (MSP) and Multi-Point Evaluation (MPE)

The next needed notion is the so-called Minimal Subspace Polynomial which is the linearized polynomial of minimal degree which vanishes over a subspace of $\mathbb{F}_2^m$. Its existence and uniqueness is ensured by the following lemma.

Lemma 1 (Minimal Subspace Polynomial (MSP), (Lidl and Niederreiter, 1997))**.** *Let $\mathcal{U}$ be a linear subspace of $\mathbb{F}_2^m$. Then there exists a unique nonzero monic polynomial $\mathcal{M}_\mathcal{U} \in \mathbb{L}_{2^m}$ of minimal degree such that* $\ker \mathcal{M}_\mathcal{U} = \mathcal{U}$*. Its degree is* $\dim \mathcal{U}$*.*

Algorithm 7: RightLEEA.

input : $a, b \in \mathbb{L}_{2^m}[x]$ with $\deg_2(a) \geq \deg_2(b)$, stopping degree d_{stop}
output : $\tau, z, y \in \mathbb{L}_{2^m}[x]$ such that $\tau = z \circ a + y \circ b$ and $\deg_2(\tau) < d_{stop}$
initialize: $z = 0, \quad y = x, \quad v = 0, \quad u = x, \quad a' = a, \quad b' = b$

while $\deg_2(b) \geq d_{stop}$ *and* $a \neq 0$ **do**
  $q, r = \text{RightDivision}(b', a')$
  $l = z + q \circ u$
  $i = y + q \circ v$
  $b' = a', \quad a' = r$
  $z = u, \quad y = v$
  $u = l, \quad v = i$
$\tau = b'$
return τ, z, y

Algorithm 8 is used to find the MSP of a subspace of $\mathbb{F}_2^m$ and requires another operation called Multi-Point Evaluation (MPE) which outputs the evaluation of a linearized polynomial over a set of points in $\mathbb{F}_{2^m}$. The algorithm to compute MPE is presented in Algorithm 9. Both algorithms apply the divide and conquer strategy and call each other recursively.

Algorithm 8: MSP.

input : Generating set $U = u_1, \ldots, u_s$ of a subspace $\mathcal{U} \in \mathbb{F}_2^m$
output: $p \in \mathbb{L}_{2^m}[x]_{\leq s}$ such that $p(u_i) = 0$ for each $i \in \{1, \ldots, s\}$

if $s = 1$ **then**
  if $U[1] = 0$ **then**
    return $p = x$
  else
    return $p = x - U[1]$
else
  for *i from* 1 *to* $\lfloor s/2 \rfloor$ **do**
    $A[i] := U[i]$
  for *i from* $\lfloor s/2 \rfloor + 1$ *to* s **do**
    $B[i - \lfloor s/2 \rfloor] = U[i]$
  $a = \text{MSP}(A)$
  $P' = \text{MPE}(a, B)$
  $b = \text{MSP}(P')$
  return $p = b \circ a$

Algorithm 9: MPE.

input : $a \in \mathbb{L}_{2^m}[x]_{\leq s}, \{u_1, \ldots, u_s\} \in \mathbb{F}_{2^m}^s$
output: Vector $A = [a(u_1), \ldots, a(u_s)] \in \mathbb{F}_{2^m}^s$

if $a = 0$ **then**
  return $A = [0_1, \ldots, 0_s]$;
if $s = 1$ **then**
  $\sigma = 0$
  for *i from* 0 *to* $\deg_2(a)$ **do**
    $\sigma = \sigma + a_{i+1} \cdot Frobenius(U[1], i)$
  return $A = [\sigma]$
else
  for *i from* 1 *to* $\lfloor s/2 \rfloor$ **do**
    $A[i] = U[i]$
  for *i from* $\lfloor s/2 \rfloor + 1$ *to* s **do**
    $B[i - \lfloor s/2 \rfloor] = U[i]$
  $w = \text{MSP}(A)$
  $w' = \text{MSP}(B)$
  $q, r = \text{RightDivision}(a, w)$
  $q', ' = \text{RightDivision}(a, w')$
  return $A = \text{MPE}(r', A) \| \text{MPE}(r', B)$

5.3.6 Interpolation

The last subroutine is shown in Algorithm 10, and depicts an instantiation of the interpolation algorithm optimized for linearized polynomials which also uses the MSP and MPE Algorithms 8, 9.

Algorithm 10: Interpolation.

input : $(w_1, y_1), \ldots, (w_s, y_s) \in \mathbb{F}_{2^m}^2$, x_i linearly independent over $\mathbb{F}_2$
output: Interpolation polynomial p such that $p(w_i) = y_i$ for all $i \in \{1, \ldots, s\}$

if $s = 1$ **then**
  return $[y[1]/w[1]]$
else
  for *i from* 1 *to* $\lfloor s/2 \rfloor$ **do**
    $A[i] = w[i]$
  for *i from* $\lfloor s/2 \rfloor + 1$ *to* s **do**
    $B[i - \lfloor s/2 \rfloor] = w[i]$
  $p = \text{MSP}(A)$
  $p' = \text{MSP}(B)$
  $Z = \text{MPE}(p', A)$
  $Z' = \text{MPE}(p, B)$
  $I = \text{Interpolation}(Z, [y[1], \ldots, y[\#Z]])$
  $I' = \text{Interpolation}(Z', [y[\#Z], \ldots, y[\#Z + \#Z']])$
  return $I \circ M' + I' \circ M$

5.3.7 Gabidulin Decoding

Finally, we present the main decoding procedure in Algorithm 11. One can immediately spot that the second step of this procedure is only depending on the generators of the chosen Gabidulin code which is part of the private key. Hence, one can store the $\text{MSP}(\langle g_i, \ldots, g_n \rangle)$ into the private key to optimize the decoding speed at the price of memory consumption. Notice that, the value v computed in the RightLEEA step is the error span polynomial as defined in (Wachter-Zeh, 2013).

Algorithm 11: Gabidulin Decoding.

input : Received word $r \in \mathbb{F}_{2^m}^n$ and the $Gab_{k,n}$ code generators $\{g_1, g_2, \ldots, g_n\}$
output: Corrected codeword z, or a decoding failure message

$\tilde{r} = \text{Interpolation}(g, r)$
$M = \text{MSP}(g)$
$r, u, v = \text{RightLEEA}(M, \tilde{r}, \lfloor (n+k)/2 \rfloor)$
$z, p = \text{LeftDivision}(r, v)$
if $p = 0$ **then**
  return z
else
  return "Decoding Failure"

6 PERFORMANCE RESULTS

Our software implementation has been designed as an extension of the Open Quantum Safe (OQS) project (Mosca et al., 2017). The main motivation was to be able to compare our scheme with the other available cryptosystems under the same conditions. We have integrated DRANKULA in the OQS library and have performed the automated benchmarks. The benchmarks have been run on a MacBook Pro 2017, 2.9 GHz Intel Core i7, 16 GB 2133 MHz LPDDR3.

Table 2 presents the OQS benchmark results. As can be observed, DRANKULA has a faster key generation than McBits as well as a significantly lower public key size. Moreover, our scheme is faster than SIDH for encryption and decryption, but that comes at the price of a larger ciphertext. Same results apply to SIKE. Although RLWE BCNS15 has the best key generation performance, its encryption is substantially worse than DRANKULA with respect to both performance and ciphertext size.

Tables 3 and Table 4 show DRANKULA's performance for the three levels of security considered

Table 2: Performance comparison between different post-quantum cryptography schemes using OQS.

Scheme	Class. sec.	PQ sec.	operation	mean CPU cycles	Bytes comm.
McBits	128	-	Key gen.	196,718,429	311,736
			Encryption	72,952	141
			Decryption	331,282	
DRANKULA	128	78	Key gen.	159,808,905	7,936
			Encryption	115,975	504
			Decryption	7,761,451	
SIDH	126	84	Key gen.	70,672,623	378
			Encryption	143,629,801	378
			Decryption	56,956,858	
SIKE	126	84	Key gen.	77,602,053	378
			Encryption	126,294,194	402
			Decryption	134,851,945	
RLWE BCNS15	163	76	Key gen.	1,807,319	4,096
			Encryption	2,895,648	4,224
			Decryption	297,673	

in this work, and for an implementation without and with carry-less multiplication optimization, respectively. As observed, a simple software optimization as is the carry-less multiplication provides a significant performance improvement.

7 A NOTE ON A SIDE-CHANNEL RESISTANT IMPLEMENTATION

When implementing a new cryptosystem there is always the challenge of finding the fastest way to compute each algorithm. As we detailed in the previous sections, the approach to implement the operations required in each algorithm could be done in different manners. A typical technique to speed up an implementation is to remove operations under certain conditions or to compute the same in a different way depending on the inputs (i.e., by means of branching). This conditional execution is usually exploited by side-channel attackers who try to gather information about the inputs of operations by studying the time difference or any other physical measurement that may lead to some information leakage.

The fact that an implementation is weak against some side-channel attack does not mean that the cryptosystem itself is not secure. It usually means that some performance improvements must be modified from the implementation so that secret information remains secure. Nevertheless, nowadays performance is such an important property for cryptosystems that the lack of an efficient implementation may end up with the cryptosystem becoming not used at all and eventually obsolete.

One alternative to protect an implementation from side-channel attacks is by means of constant-time operations. This way, the operations required by each algorithm take the same time no matter the input and, therefore, an attacker can only extract information about the operations, which might be public, but not about the inputs and outputs. The main drawback of this approach is that the cost of the algorithms is heavily increased. For this reason, we propose to modify only a subset of operations to be constant-time, and we analyze the information leaked to check the feasibility of obtaining any knowledge of the secret key, or the error generated during encryption, from the leaked data.

We propose to only modify some algorithms used for the decryption procedure in order to make them constant time, namely: the linearized polynomial addition, the linearized polynomial multiplication and part of the Minimal Subspace Polynomial. The first modification is simple, we just computed the addition for all the elements and checked the degree by iterating over all the positions. For the multiplication, in Algorithm 12 we modified a little bit the previous Algorithm 4 to avoid conditional instructions in the loop,. We also computed the multiplication up to the maximum degree, not only up to the degree of the polynomials.

Finally, the changes in MSP only affected the initial conditions to return values. In Algorithm 8 we returned either 1 or 2 values depending if the input subspace is 0 or not. This would leak information of the 0s in the input subspace. To prevent that from happening we generated the same result but we fill the polynomial with a 0 in the second position. More precisely, we change the return value so instead of the return statement in Algorithm 8 line 4, we return $p = x - 0$.

Table 3: Performance comparison for different security levels without carry-less multiplication.

Scheme	Class. sec.	PQ sec.	operation	mean CPU cycles	Bytes comm.
DRANKULA	128	78	Key gen.	717,478,286	7,936
			Encryption	1,167,473	504
			Decryption	64,645,692	
DRANKULA	192	104	Key gen.	1,405,999,903	15,120
			Encryption	2,154,776	852
			Decryption	131,834,096	
DRANKULA	256	139	Key gen.	3,537,521,289	27,648
			Encryption	3,887,225	1,152
			Decryption	276,356,299	

Table 4: Performance comparison for different security levels with carry-less multiplication.

Scheme	Class. sec.	PQ sec.	operation	mean CPU cycles	Bytes
DRANKULA	128	78	Key gen.	159,808,905	7,936
			Encryption	115,975	504
			Decryption	7,761,451	
DRANKULA	192	104	Key gen.	281,555,550	15,120
			Encryption	209,117	852
			Decryption	127,45,290	
DRANKULA	256	139	Key gen.	544,649,920	27,648
			Encryption	320,951	1,152
			Decryption	23,559,649	

Algorithm 12: Constant-Time Multiplication.

input : $a, b \in \mathbb{L}_{2^m}[x]_{\leq n+1}$
output: $c = a \circ b \in \mathbb{L}_{2^m}[x]_{\leq n+1}$

for *i from* 0 *to* $n+1$ **do**
  $y = 0$
  for *j from* 0 *to* *i* **do**
    $z = \text{Frobenius}(b_{i-j}, j)$
    $z = z \cdot a_j$
    if $j \leq \deg_2(a)$ *and* $i - j \leq \deg_2(b)$ **then**
      $y = y + z$
    else
      $y = y + 0$
  $c_i = y$
return c

We have chosen only these three operations because our analysis showed that with these changes, the decryption would not leak not enough information to threaten the security of the cryptosystem. Indeed, the unmodified operations are: interpolation, rightLEEA and left / right division, which usually take polynomials of the same degree as inputs at a given recursion degree.

Table 5 shows the performance of our constant-time implementation of DRANKULA for the same three security levels that we have considered in our work. It must be noted that our solution uses the carry-less multiplication and, as the constant-time implementation only affects the decryption, we skip the other two operations. Results show that decryption is heavily impacted when the countermeasures are put in place.

The only identified cases where an attacker would observe a timing difference are the following:

1. The interpolation polynomial $\tilde{r}$ is of 2-degree less than $n-1$: the attacker would be able to learn that the point (g_n, r_n) is already part of the graph of the interpolation polynomial of $\{(g_i, r_i)\}_{1 \leq i < n}$. As the attacker does not have any control on the g_i and neither on the r_is (as they are actually the coefficients of cP where P is unknown to him), he will not be able to extract any meaningful information.
2. The degree controlling the number of iterations in the loop of RightLEEA or the right division is decreasing more than 1: the attacker would observe in the best case that the Interpolation polynomial $\tilde{r}$ divides the MSP of the generators of the Gabidulin code. Again, as the coefficients of $\tilde{r}$ are actually depending on the coefficients of cP, then the attacker will not gain any valuable knowledge on the private key.

Nevertheless, a deeper analysis of the possible leakages through timing analysis constitutes a path for future works.

Table 5: Performance comparison for different security levels of the constant-time implementation with carry-less multiplication.

Scheme	Class. sec.	PQ sec.	operation	mean CPU cycles
DRANKULA	128	78	Decryption	314,116,991
DRANKULA	192	104	Decryption	878,204,550
DRANKULA	256	139	Decryption	2,662,744,720

8 CONCLUSIONS AND FUTURE WORK

This work is presenting a software implementation of DRANKULA, a rank based McEliece-like cryptosystem with deterministic decryption presented in (Loidreau, 2017), and its performance results. We address several caveats of the scheme when carrying out a practical implementation, and we provide three sets parameters targeting 64, 96 and 128 bits of post-quantum security. In addition we provide the pseudocode for the main subroutines of our algorithms, which might be helpful to the community to continue investigating this scheme. Results show that DRANKULA is a viable alternative to other post-quantum cryptography schemes and efficient in terms of key sizes and computational complexity. We end up providing a note on a side-channel resistant implementation of our proposal. As future work it would be interesting to formally investigate the IND-CCA and IND-CPA properties of DRANKULA.

REFERENCES

Aragon, N., Blazy, O., Deneuville, J.-C., Gaborit, P., Hauteville, A., Ruatta, O., Tillich, J.-P., and Zemor, G. (2017). Locker - low rank parity check codes encryption.

Bernstein, D. J., Buchmann, J., and Dahmen, E. (2008). *Post Quantum Cryptography*. Springer Publishing Company, Incorporated, 1st edition.

Bernstein, D. J., Chou, T., and Schwabe, P. (2015). Mcbits: fast constant-time code-based cryptography. *IACR Cryptology ePrint Archive*, 2015:610.

Bosma, W., Cannon, J., and Playoust, C. (1997). The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265. Computational algebra and number theory (London, 1993).

Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., and Smith-Tone, D. (2016). Report on post-quantum cryptography.

Chou, T. (2016). Qcbits: Constant-time small-key code-based cryptography. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 280–300.

Chou, T. (2017). Mcbits revisited. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 213–231. Springer.

Deneuville, J.-C., Gaborit, P., and Zémor, G. (2017). Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In *International Workshop on Post-Quantum Cryptography*, pages 18–34. Springer.

Devoret, M. H. and Schoelkopf, R. J. (2013). Superconducting circuits for quantum information: An outlook. *Science*, 339(6124):1169–1174.

Gabidulin, E. M. (1985). Theory of codes with maximum rank distance. *Problems of Information Transmission (English translation of Problemy Peredachi Informatsii)*, 21(1).

Gabidulin, E. M., Paramonov, A. V., and Tretjakov, O. V. (1991). *Ideals over a Non-Commutative Ring and their Application in Cryptology*, pages 482–489.

Gaborit, P. (2005). *Shorter keys for code based cryptography*, pages 81–90.

Gaborit, P., Murat, G., Ruatta, O., and Zmor, G. (2013). Low rank parity check codes and their application to cryptography.

Gao, S. (2003). *A New Algorithm for Decoding Reed-Solomon Codes*, pages 55–68.

Google (2018). A preview of bristlecone, googles new quantum processor. Available at https://research.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html.

Gueron, S. and Kounavis, M. E. (2010). Intel® carry-less multiplication instruction and its usage for computing the gcm mode. *White Paper*.

Hankerson, D., Menezes, A. J., and Vanstone, S. (2006). *Guide to elliptic curve cryptography*. Springer Science & Business Media.

Itoh, T. and Tsujii, S. (1988). A fast algorithm for computing multiplicative inverses in gf (2m) using normal bases. *Information and computation*, 78(3):171–177.

Karatsuba, A. and Ofman, Y. (1962). Multiplication of many-digital numbers by automatic computers. *Doklady Akademii Nauk SSSR, Translation in Physics-Doklady 7, 595-596, 1963*, 145(2):293–294.

Lidl, R. and Niederreiter, H. (1997). *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition.

Loidreau, P. (2017). *A New Rank Metric Codes Based Encryption Scheme*, pages 3–17.

Maples, K. (2013). Singularity of random matrices over finite fields.

McEliece, R. J. (1978). A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116.

Melchor, C. A., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.-C., Gaborit, P., and Zmor, G. (2016). Rank quasi-cyclic (rqc).

Misoczki, R., Tillich, J., Sendrier, N., and Barreto, P. S. L. M. (2013). Mdpc-mceliece: New mceliece variants from moderate density parity-check codes. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 2069–2073.

Mosca, M., Stebila, D., and Contributors (2017). Open quantum safe.

Niederreiter, H. (1986). Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15:159–166.

NIST (2018). Round 1 submissions. Available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.

Otmani, A., Kalachi, H. T., and Ndjeya, S. (2016). Improved cryptanalysis of rank metric schemes based on gabidulin codes. *CoRR*, abs/1602.08549.

Overbeck, R. (2005). *A New Structural Attack for GPT and Variants*, pages 50–63.

Overbeck, R. (2008). Structural attacks for public-key cryptosystems based on gabidulin codes. *Journal of Cryptology*, 21(2):280–301.

Puchinger, S. and Wachter-Zeh, A. (2015). Fast operations on linearized polynomials and their applications in coding theory. *CoRR*, abs/1512.06520.

Sendrier, N. (2017). Code-based cryptography: State of the art and perspectives. *IEEE Security & Privacy*, 15(4):44–50.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509.

Swan, R. G. (1962). Vector bundles and projective modules. *Transactions of the American Mathematical Society*, 105(2):264–277.

Wachter-Zeh, A. (2013). *Decoding of block and convolutional codes in rank metric, PhD thesis*.